Statistical Techniques in Robotics (16-831, S21) Lecture #18 (Monday, April 12)

Model-Free Value Prediction (Part 2)

Lecturer: Kris Kitani *Scribes: Zach Patterson, Samuel Triest*

1 Review

Last lecture introduced the concept of model-free value prediction, or the process of inferring a value function $V(s)$ without access to the underlying MDP (i.e. no $p(s'|s,a)$ or $r(s,a,s')$). In order to compute a value function, we instead use interaction samples from the target environment . This extension of the framework is often important when considering systems that act in real-world environments, as it is often impractical/impossible to write down a transition function that perfectly captures the true dynamics of the system.

1.1 Problem Setup

The current problem formulation is as follows:

Goal: Estimate the value function $v_\pi(s)$ of a policy π acting in an environment $\mathcal{E}$ that is "close" [1] to the true value function $V_\pi(S)$ of the π in $\mathcal{E}$.

Inputs: We are given the policy $\pi(\cdot|s)$ and a set of K trajectories τ, where τ_k is defined as follows:

$\tau_k = [(s_1, a_1, r_1, s_2), \dots (s_{T-1}, a_{T-1}, r_{T-1}, s_T)]$, where:

- $s_1 \sim \rho(\cdot)$
- $a_t \sim \pi(\cdot|s_t) \forall t$
- $r_t \sim r(s_t, a_t, s_{t+1})) \forall t < T$
- $s_t \sim p(\cdot|a_{t-1}, s_{t-1}) \forall t > 1$

Essentially, we are given k trajectories τ, where each trajectory is comprised of T consecutive interactions in the environment by the policy. Each tuple in the trajectory records how the state changed when the policy took action a in state s, and how much reward this transition incurred. Note that $\rho(\cdot)$, $(\cdot|a,s)$ and $r(s_t, a_t, s_{t+1})$ are all part of the environment and are not available to our value prediction algorithm. Instead, we must infer (usually implicitly) these distributions from out data $\tau_1 \dots \tau_k$. This is the major difference between model-based and model-free value prediction.

[1] I'm using a vague term here because as mentioned in class, different estimators minimize different measures of closeness (i.e. MSE vs. MLE).

1.2 Monte-Carlo Prediction

In general, in order to perform model-free value prediction, we need to use some estimator of the value function. This estimator will be referred to as $G^{(t)}$. The previous lecture introduced an algorithm that uses an intuitive estimator $G^{(t)}$: the Monte-Carlo Estimator. Given a trajectory τ, the Monte-Carlo estimator says that:

$$G^{(t)} \triangleq \Sigma_{k=t}^{T} \gamma^{k-t} r_k \tag{1}$$

Simply put, the Monte-Carlo estimator says that we can estimate $V_\pi(s)$ as our observed return from state s. Using this estimator, we can derive several algorithms for model-free value prediction, with the most straightforward being every-visit MC prediction, presented in Algorithm 4.

Algorithm 1 Every-Visit MC Prdiction

1: $V(s) \leftarrow k \ \ \forall s \in \mathcal{S}$ ▷ Initialize the value function to some arbitrary value.
2: **for** $1 \ \ldots \ K$ **do**
3: $\quad \tau_k \sim \pi, \mathcal{E}$ ▷ Sample a trajectory by rolling out the policy in the environment.
4: $\quad s_t, a_t, r_t, s_{t+1} \sim \tau_k$ ▷ Decompose trajectory into transitions.
5: $\quad$ **for** $t \in 0 \ldots T$ **do**
6: $\quad\quad G^{(t)} = \Sigma_{i=t}^{T} \gamma^{i-t} r_i$ ▷ Estimate V as reward-to-go from current state
7: $\quad\quad V(s_t) \leftarrow V(s_t) + \alpha(G^{(t)} - V(s_t))$ ▷ Update V by taking a step towards the estimator $G^{(t)}$
8: $\quad$ **end for**
9: **end for**

One interesting thing to note is that this algorithm can be considered as a form of online gradient descent (see line 7), where the objective function that we are trying to minimize is:

$$J(\theta) = \frac{\alpha}{2}(G^{(t)} - V(s_t))^2 \tag{2}$$

We can observe that this is simply a scaled version of the MSE between our value function and our value estimator. This algorithm has several properties that will be discussed later in these notes, as to directly compare them with TD Prediction.

2 Summary

2.1 Temporal Difference (TD) Prediction

Outline: Derive estimator from Bellman eq. Talk about 1-step vs. n-step. Mention MC as TD-inf. Give 1 step vs. n-step algo. Discuss tradeoffs: bias/variance. Usability online. MSE vs. MLe (cite the Sutton paper). Segway: What is optimal n?

Using the recursive definition of the V function as:

$$V_\pi(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s), s' \sim p(\cdot|a,s)} [r(s, a, s') + \gamma V_\pi(s')] \tag{3}$$

we can derive another family of potentially useful estimators of V_π. Since the data we've collected can be assumed to come from the policy π and dynamics p, we can construct an estimator:

$$G^{(t)} \triangleq r_t + \gamma V_\pi(s_{t+1}) \tag{4}$$

This estimator is known as the TD(0) estimator, or the 1-step TD Estimator. The recursive nature of the Bellman equation lets us derive a family of estimators known as n-step TD estimators, where the n-step TD estimator $G^{(t)}(n)$ is:

$$G^{(t)}(n) \triangleq \sum_{k=t}^{t+n} [\gamma^{k-t} r_k] + \gamma^n V_\pi(s_{t+n}) \tag{5}$$

In general, the TD estimators utilize a technique called bootstrapping, which refers to how the definition of the estimator for a given state requires using the estimator on other states (with the base case being at terminal states).

Another interesting fact is that the Monte-Carlo estimator is equal to the ∞-step TD estimator. This follows from the fact that $\gamma^\infty = 0, \forall \gamma \in [0, 1)$. As such, we are left with the discounted sum of rewards for all timesteps (which is just the Monte-Carlo estimator).

Below are the algorithms for model-free value prediction using the 1-step and n-step TD estimators.

Algorithm 2 1-Step TD Prediction

$V(s) \leftarrow k \ \ \forall s \in \mathcal{S}$ ▷ Initialize the value function to some arbitrary value.
for $1 \ \ldots \ K$ **do**
  $\tau_k \sim \pi, \mathcal{E}$ ▷ Sample a trajectory by rolling out the policy in the environment.
  $s_t, a_t, r_t, s_{t+1} \sim \tau_k$ ▷ Decompose trajectory into transitions.
  for $t \in 0 \ldots T-1$ **do**
    $G^{(t)} = r_t + \gamma V(s_{t+1})$ ▷ Estimate V using 1-step TD.
    $V(s_t) \leftarrow V(s_t) + \alpha(G^{(t)} - V(s_t))$ ▷ Update V by taking a step towards the estimator $G^{(t)}$
  end for
  $G^{(T)} = r_T$ ▷ Handle the final state.
  $V(s_T) \leftarrow V(s_T) + \alpha(G^{(T)} - V(s_T))$
end for

Algorithm 3 n-Step TD Prediction

$V(s) \leftarrow k \ \ \forall s \in \mathcal{S}$ ▷ Initialize the value function to some arbitrary value.
for $1 \ \ldots \ K$ **do**
  $\tau_k \sim \pi, \mathcal{E}$ ▷ Sample a trajectory by rolling out the policy in the environment.
  $s_t, a_t, r_t, s_{t+1} \sim \tau_k$ ▷ Decompose trajectory into transitions.
  for $t \in 0 \ldots T - n$ **do**
    $G^{(t)} = \sum_{i=t}^{t+n} [\gamma^{i-t} r_i] + \gamma^n V(s_{t+n})$ ▷ Estimate V using n-step TD.
    $V(s_t) \leftarrow V(s_t) + \alpha(G^{(t)} - V(s_t))$ ▷ Update V by taking a step towards the estimator $G^{(t)}$
  end for
  for $t \in (T - n + 1) \ldots T$ **do**
    $G^{(t)} = \sum_{i=t}^{T} [\gamma^{i-t} r_i] + \gamma^{T-n} V(s_T)$ ▷ Handle the last n states.
    $V(s_t) \leftarrow V(s_t) + \alpha(G^{(t)} - V(s_t))$
  end for
end for

As mentioned, there are several tradeoffs between TD estimators and Monte-Carlo estimators. Some important tradeoffs to consider are:

1. Usability Online
2. Bias/Variance Tradeoff

The following subsections will discuss the tradeoffs between the Monte-Carlo estimator and the 1-step TD estimator as they present extreme points on the tradeoff spectrum, with the n-step TD estimators falling some place in the middle.

2.1.1 Usability Online

One major distinction between the Monte-Carlo estimator and the 1-step TD estimator is that we cannot use the Monte-Carlo estimator in an online fashion. Recall the Monte-Carlo estimator:

$$G^{(t)} \triangleq \Sigma_{k=t}^{T} \gamma^{k-t} r_k$$

As we can see, in order to get the estimate at time t, we require the reward at the final timestep T. This is not the case for the 1-step (and n-step) TD estimator:

$$G^{(t)} \triangleq r_t + \gamma V_\pi(s_{t+1})$$

where we only require the reward at the current timestep and the next state (for n-step TD, we need the next n rewards and the state at t_n).

2.1.2 Bias/Variance Tradeoff

The Monte-Carlo estimator and 1-step TD estimator have a very significant tradeoff between their bias and variance. In general, the Monte-Carlo estimator has high variance but no bias, while the

1-step TD estimator has low variance but nonzero bias. Recall that the definition of bias of an estimator G of a quantity x is:

$$B(G) = \mathbb{E}[G] - x \tag{6}$$

For our case, the quantity that we try to measure is $v_\pi(s) = \mathbb{E}_{\tau\sim\pi,\mathcal{E}}[\sum_t \gamma^t r_t]$ (using lowercase to indicate that this is the true value function and not our estimator $V_\pi(s)$). Clearly, the Monte-Carlo estimator is unbiased:

$$B(G_{MC}) = \mathop{\mathbb{E}}_{\tau\sim\pi,\mathcal{E}}[\sum_t \gamma^t r_t] - \mathop{\mathbb{E}}_{\tau\sim\pi,\mathcal{E}}[\sum_t \gamma^t r_t] = 0 \tag{7}$$

However, the bias of the 1-step TD-estimator is not necessarily 0.

$$B(G_{TD}) = \mathop{\mathbb{E}}_{\tau\sim\pi,\mathcal{E}}[r_t + \gamma V(s_{t+1})] - \mathop{\mathbb{E}}_{\tau\sim\pi,\mathcal{E}}[\sum_t \gamma^t r_t] = \mathop{\mathbb{E}}_{\tau\sim\pi,\mathcal{E}}[\gamma V(s_{t+1}) - \sum_{i=t+1}^{T} \gamma^i r_i] \tag{8}$$

Note that this V in the bias is not the true v and can take an arbitrary value depending on its initialization.

On the other hand, the Monte-Carlo estimator has much higher variance due to the fact that it contains many more terms, which each have variance.

Overall, the decision on whether to use Monte-Carlo estimates or TD estimates is problem-specific. However, the authors note that TD estimation seems to be more popular, as can be observed by its use to estimate value functions in many seminal papers in RL in recent years. [1, 2, 3, 4, 5] all use 1-step TD to train a neural network estimator of either the V or Q function, and [6] uses an n-step TD estimator.

2.2 TD-λ

2.2.1 Offline- λ-Return

At the end of the last section, we saw that the success of the TD algorithm is dependent on the number of steps ahead used in the estimate (N). In this section, we'll discuss an algorithm that allows us to avoid specifying N. At the heart of this approach, called TD(λ), is the realization that we can combine multiple estimates to form a TD Target, for example:

$$G^{(t)} = \frac{1}{3}G^{(t)}(3) + \frac{1}{3}G^{(t)}(5)\frac{1}{3}G^{(t)}(7). \tag{9}$$

So what if we combine the targets for all N? This gives us the infinite horizon λ-Return:

$$G_\lambda^{(t)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G^t(n), \lambda \in [0,1]. \tag{10}$$

λ, called the trace decay factor, provides a weighting scheme in which the longer estimates (higher n) are given less weight. See 1 for a depiction of this effect.

In practice, we must truncate the return to a finite horizon case:

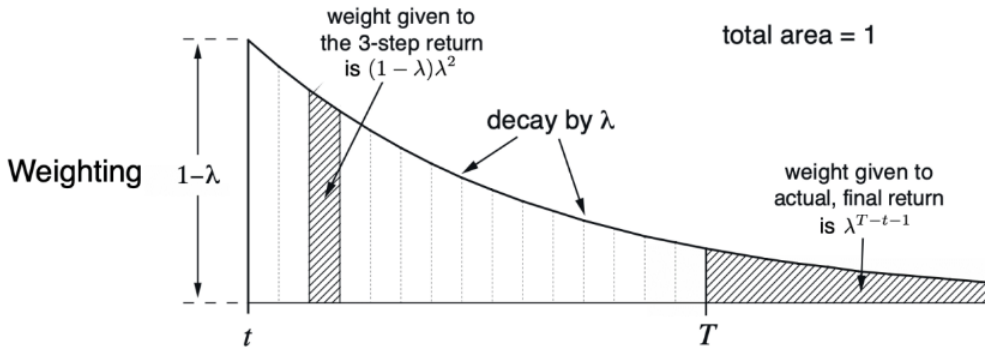

Figure 1: Shows the decay of the λ-Return with increasing steps [7].

$$G_\lambda^{(t)} = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G^t(n) + \lambda^{T-t-1}G^t(T-t-1), \lambda \in [0,1]. \tag{11}$$

Looking at a few special cases, we see that when $\lambda = 1$, $G_\lambda^{(t)} = G^t(T-t-1)$, which is the Monte Carlo estimate. When $\lambda = 0$, $G_\lambda^{(t)} = G^t(1)$, which is the one step TD estimate. The following shows the Offline-λ-Return algorithm:

Algorithm 4 Offline-λ-Return(π, α, λ)

for $1 \ldots E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi$ ▷ Sample a full episode.
  for $t \in 0 \ddots T$ **do**
    $G_\lambda^{(t)} \leftarrow (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G^{(t)}(n)$ ▷ Precompute all returns.
  end for
  for $t \in 0 \ddots T$ **do**
    $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha[G_\lambda^{(t)} - V(s^{(t)})]$ ▷ Precompute all returns.
  end for
end for

Figure 2 shows that the offline lambda return algorithm is only a bit better than the n-step TD approach so why would we use it? The answer is that this algorithm provides a tool for understanding λ-returns but it is not a practical algorithm because you need to access the full future trajectory to update the value function (impossible online). However, the 'backwards TD(λ) perspective' is useful for the practical implementation, which uses the concept of eligibility traces.

2.2.2 Eligibility Returns

The TD(λ) update can be written as a sum of weighted one step TD errors:

$$\Delta V(s^t) = G_\lambda^{(t)} - V(s^t) = \sum_{i=0}^{\infty}(\gamma\lambda)^i\delta^{(t+i)} \tag{12}$$

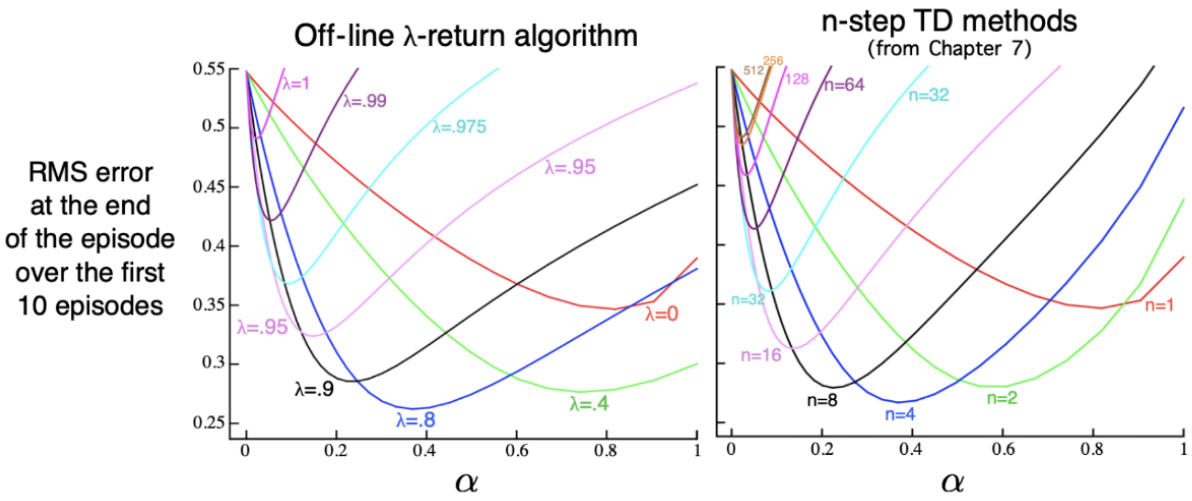

Figure 2: Shows the performance of Offline-λ-Return versus the n-step TD [7].

$$\delta^{(t)} = r(t) + \gamma V(s^{(t+1)}) - V(s^{(t)}). \tag{13}$$

It's important to note that λ and γ are not subsumed into the same discount factor because they will eventually come to have independent effects on the function. The above is easy to prove algebraically by plugging in the definition of λ-return, expanding the sum, separating out the n-step TD targets, and canceling terms.

We can equivalently write the update as a decaying weighted sum of future TD errors at time k:

$$\Delta V(s^{(k)} = s) = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta^{(k+i)} = \sum_{t=k}^{\infty} (\gamma\lambda)^{t-k} \delta^{(t)}. \tag{14}$$

We can now rewrite this as:

$$\Delta V(s^{(k)} = s) = \sum_{t=k}^{\infty} (\gamma\lambda)^{t-k} \delta^{(t)} = \sum_{t=1}^{k-1} (0)\delta^{(t)} + \sum_{t=k}^{\infty} (\gamma\lambda)^{t-k} \delta^{(t)} = \sum_{t=1}^{\infty} z^{(t)}(s)\delta^{(t)} \tag{15}$$

where

$$z^{(t)}(s) = \begin{cases} 0 & t < k \\ (\gamma\lambda)^{t-k} & t \geq k. \end{cases} \tag{16}$$

This new value function update expression contains the weight/credit function z within the infinite horizon sum, giving no weight to the past and a decaying weight to the future. This weighting/credit function is called the eligibility trace.
Therefore, the eligibility trace is 0 prior to the occurrence of some state s at k. When t=k, the eligibility trace is 1 and it decays after time k with time.
The eligibility trace can also be written as an incremental update:

$$z^{(t)}(s) = \gamma\lambda z^{(t-1)}(s) + \mathbb{1}[s^{(t)} = s]. \tag{17}$$

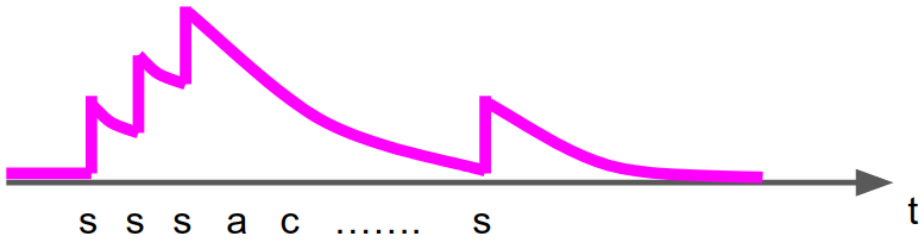

Figure 3: The eligibility trace over time. The x axis is time. Note that states are written when they occur as letters. In this case, we're tracking the eligibility trace of state s.

This form is slightly different than the previous form in that every time it sees state s, 1 is added (as opposed to being set to 1 when seeing state s). Then the function decays according to the same discount factor and trace decay factor as before. This gives additional weight to frequency as well as the to recency. See Figure 3 for a depiction of this function.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[6] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2020.